Modelo para classificação de widgets do tipo Dropdown e Suggestion Box ¶

Importando bibliotecas

In [1]:

```
# https://docs.python.org/3/library/random.html
import random

# funções e operações para cálculos numéricos / https://docs.scipy.org/doc/numpy/reference/
import numpy as np
print("numpy version: {}". format(np.__version__))

# manipulação e processamento de dados / https://pandas.pydata.org/pandas-docs/stable/
import pandas as pd
print("pandas version: {}". format(pd.__version__))

# pré processamento de dados / https://scikit-learn.org/stable/modules/preprocessing.html
import sklearn.preprocessing as preprocessing

#ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
numpy version: 1.16.4
pandas version: 0.25.0
```

In [2]:

```
path_file1 = 'DataSets/widgets/5-test-widget-com-url.tab'
file1 = pd.read_csv(path_file1, sep='\t')

path_file2 = 'DataSets/widgets/5-training-widget-com-url.tab'
file2 = pd.read_csv(path_file2, sep='\t')

frames = [file1, file2]
dataset = pd.concat(frames)

dataset = dataset[dataset.widgetClass != 'Discard']

dataset.head(10)
```

Out[2]:

	positionLeft	positionTop	width	height	childCount	domHeight	domLevelMoreElements	
0	1119	16	248	30	175	7	6	
1	1263	16	30	30	0	0	1	
2	0	0	0	0	156	4	3	
3	0	0	0	0	0	0	1	
4	1293	16	45	30	38	6	1	
5	1308	16	30	30	1	1	1	
6	0	0	0	0	32	5	1	
7	1263	16	30	30	163	5	4	
8	1017	613	320	40	0	0	1	
9	0	0	0	0	2	1	1	

In [3]:

```
dataset.describe()
```

Out[3]:

	positionLeft	positionTop	width	height	childCount	domHeig
count	43347.000000	4.334700e+04	43347.000000	43347.000000	43347.000000	43347.0000
mean	-1152.049554	-8.698751e+02	269.255612	67.907006	13.469098	2.5286
std	38719.137384	3.876365e+04	342.143788	164.844927	27.460319	2.8336
min	-1000000.000000	-1.002171e+06	0.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.0000
50%	370.000000	1.660000e+02	139.000000	22.000000	2.000000	1.0000
75%	684.000000	5.380000e+02	440.000000	46.500000	11.000000	4.0000
max	2702.000000	1.991900e+04	6755.000000	3824.000000	235.000000	17.0000

In [4]:

```
dataset['widgetClass'].value_counts()
```

Out[4]:

```
Other            33478
Dropdown          6459
Suggestionbox     3410
Name: widgetClass, dtype: int64
```

In [5]:

```
dataset.loc[(dataset['widgetClass'] == 'Dropdown'), 'widgetClass'] = 0
dataset.loc[(dataset['widgetClass'] == 'Other'), 'widgetClass'] = 1
dataset.loc[(dataset['widgetClass'] == 'Suggestionbox'), 'widgetClass'] = 2

dataset.rename(columns={'widgetClass': 'target'}, inplace=True)

dataset['target'].value_counts()
```

Out[5]:

```
1    33478
0     6459
2     3410
Name: target, dtype: int64
```

In [6]:

```
dataset['event'].value_counts()
```

Out[6]:

```
mouseover    18297
click        13691
none         10637
keyup          722
Name: event, dtype: int64
```

In [7]:

```
dataset.loc[(dataset['event'] == 'mouseover'), 'event'] = 0
dataset.loc[(dataset['event'] == 'click'), 'event'] = 1
dataset.loc[(dataset['event'] == 'none'), 'event'] = 2
dataset.loc[(dataset['event'] == 'keyup'), 'event'] = 3

dataset['event'].value_counts()
```

Out[7]:

```
0    18297
1    13691
2    10637
3      722
Name: event, dtype: int64
```

In [8]:

```
from sklearn.model_selection import cross_validate, StratifiedKFold, GridSearchCV, trai
n_test_split, GroupKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

In [9]:

```
class Classificador():
    def __init__ (self, seed):
        self._seed = seed

    def classificar(self, dataset, fold_method):
        X = dataset.loc[:, ~dataset.columns.isin(['target', 'url'])]
        y = dataset['target'] ## seleciona o "rótulo" que será classificado
        X.head(5)

        random.seed(self._seed) ## gera seed

        if(fold_method == "StratifiedKfold"):
            ## StratifiedKfold separa os folds preservando a porcentagem de amostras pa
ra cada classe do target ##
            # n_splits = número de folds
            # random_state = seed
            fold = StratifiedKFold(n_splits = 10, random_state = self._seed)

        elif(fold_method == "GroupKfold"):
            splits = len(dataset.url.unique())
            fold = list(GroupKFold(n_splits = splits).split(X, y, dataset.url))

        modelo = self.get_classificador(X, y)
        ## cv = método de cross validation
        ## scoring = métricas para avaliação do modelo
        ## f1_macro = calcula f1 (média harmônica de precision e recall) separado por c
lasses, mas sem estabelecer pesos
        scores = cross_validate(modelo, X, y, cv = fold, scoring = ['accuracy', 'f1_mac
ro', 'precision_macro', 'recall_macro'])
        print("Accuracy: %s on average and %s SD" % (scores['test_accuracy'].mean(), sc
ores['test_accuracy'].std()))
        print("Precision: %s on average and %s SD" % (scores['test_precision_macro'].me
an(), scores['test_precision_macro'].std()))
        print("Recall: %s on average and %s SD" % (scores['test_recall_macro'].mean(),
scores['test_recall_macro'].std()))
        print("F-measure: %s on average and %s SD" % (scores['test_f1_macro'].mean(), s
cores['test_f1_macro'].std()))

        return dataset

class DecisionTree(Classificador):
    def __init__ (self, seed = 42):
        Classificador.__init__(self, seed)
        self.tipo = 'decision_tree'

    def get_classificador(self, X, y):
        print('-- DECISION TREE --')
        model = DecisionTreeClassifier(random_state=self._seed)
        return model

class KNN(Classificador):
    def __init__ (self, seed = 42):
        Classificador.__init__(self, seed)
        self.tipo = 'decision_tree'

    def get_classificador(self, X, y):
        print('-- KNN --')
        model = KNeighborsClassifier()
        return model
```

```
class SVM(Classificador):
    def __init__ (self, seed = 42):
        Classificador.__init__(self, seed)
        self.tipo = 'decision_tree'

    def get_classificador(self, X, y):
        print('-- SVM --')
        model = SVC(random_state=self._seed, gamma = 'scale')
        return model

class RandomForest(Classificador):
    def __init__ (self, seed = 42):
        Classificador.__init__(self, seed)
        self.tipo = 'decision_tree'

    def get_classificador(self, X, y):
        print('-- RANDOM FOREST --')
        model = RandomForestClassifier(random_state=self._seed, n_estimators = 100)
        return model

class LR(Classificador):
    def __init__ (self, seed = 42):
        Classificador.__init__(self, seed)
        self.tipo = 'decision_tree'

    def get_classificador(self, X, y):
        print('-- LOGISTIC REGRESSION --')
        model = LogisticRegression(random_state=self._seed, multi_class = 'auto')
        return model
```

In [10]:

```
classificador = DecisionTree(42)
classificador.classificar(dataset, "GroupKfold");
```

```
-- DECISION TREE --
Accuracy: 0.9547206623024866 on average and 0.09450152340293984 SD
Precision: 0.8995573674667249 on average and 0.17211021364206888 SD
Recall: 0.8931987185428572 on average and 0.17700709662130032 SD
F-measure: 0.8858809436714605 on average and 0.1840808289006933 SD
```

In [11]:

```
classificador = KNN(42)
classificador.classificar(dataset, "GroupKfold");
```

```
-- KNN --
Accuracy: 0.7779359461527495 on average and 0.19123820700612487 SD
Precision: 0.5601618104956587 on average and 0.2293702103439111 SD
Recall: 0.5491546453322889 on average and 0.22279792748397093 SD
F-measure: 0.5291194135284941 on average and 0.22917361045193121 SD
```

In [12]:

```
classificador = SVM(42)
classificador.classificar(dataset, "GroupKfold");
```

```
-- SVM --
Accuracy: 0.7742615868162551 on average and 0.20780865821109623 SD
Precision: 0.4568205603855559 on average and 0.29129753997866464 SD
Recall: 0.5479232804232803 on average and 0.237084801484233 SD
F-measure: 0.4879082010108952 on average and 0.2706596204533156 SD
```

In [13]:

```
classificador = RandomForest(42)
classificador.classificar(dataset, "GroupKfold");
```

```
-- RANDOM FOREST --
Accuracy: 0.9656395453065041 on average and 0.06924631092564491 SD
Precision: 0.8825932838560924 on average and 0.17339488577905696 SD
Recall: 0.8691012577304325 on average and 0.18011966517714806 SD
F-measure: 0.8675389412565677 on average and 0.18275518503042798 SD
```

In [14]:

```
classificador = LR(42)
classificador.classificar(dataset, "GroupKfold");
```

```
-- LOGISTIC REGRESSION --
Accuracy: 0.879470406686373 on average and 0.12868196475462912 SD
Precision: 0.7395963582156907 on average and 0.21856065079469467 SD
Recall: 0.7065020236753874 on average and 0.21875796387039464 SD
F-measure: 0.7042761229624949 on average and 0.21776229215735082 SD
```

In [ ]:

```

```